

Constraint Library Reference

1. Constraints in NDDL

1. Temporal Constraints
2. Comparison Constraints
3. Calculation Constraints
4. Set Constraints
5. Conditional and Test Constraints
6. Object Hierarchy Constraints
7. Miscellaneous Constraints
8. Deprecated Constraint Names

Constraints in NDDL

The table that follows overviews the built-in constraints, available in NDDL or through the programmatic API. Note that:

- Many constraints can have a variable number of parameters; we use letters (a,b,etc) to indicate required parameters, and '...' to indicate that subsequent parameters are optional.
- In the *Variable Types* column, we describe restrictions on the variable types as follows:
 - ♦ **Numeric**: Type is *int*, *float*, *bool*, or some user-defined numeric type. For *bool*, the usual C/C++ convention is used; 0 \Leftrightarrow false, non-zero \Rightarrow true, and true \Rightarrow 1.
 - ♦ **Numeric Intervals**: Same as Numeric except that enumerated types (enumerated numeric types are possible) are disallowed.
 - ♦ **Comparable**: Comparable variables must all be numeric, or must all be the same type.
- We use *lb* and *ub* to refer to the lower and upper bounds, respectively, of a variable's domain.
- Maintenance of constraints means that for any value in the domain of any variable, there exists values in the domains of all other variables such that the desired property holds. For example, *eq(a,b)* means that for any x in the domain of a, there is a y in the domain of b such that $x == y$ (ie. it restricts the domains of a and b to be the intersection of those domains).

Temporal Constraints

Note that temporal constraints will in most cases provide stronger propagation than similar constraints the follow since the Temporal Network will be used. For example *precedes(a,b)* and *leq(a,b)* will enforce the same semantics; however, for the former, the Temporal Network will be able to do more propagation with other temporal constraints that involve variables a,b and detect conflicts earlier.

Constraint	Syntax	Description	Variable Types
temporalDistance	temporalDistance(a, b, c)	$a+b=c$	Numeric interval
precedes	precedes(a,b)	$a \leq b$	Numeric interval
concurrent	concurrent(a,b)	$a == b$	Numeric interval

In addition to the three basic temporal constraints, there are built-in temporal constraints for all Allen Relations ([Allen, 1983](#)) among tokens. Before listing those constraints, some distinctions are in order:

- Like most constraints on this page, the three temporal constraints above are applied to variables.
- The temporal constraints below are applied to *Tokens* (and implicitly invoke constraints between the *start* and *end* member variables of those Tokens).

- The Allen temporal constraints here should not be confused with the Allen operators (with the same names, but different syntax) described in the [NDDL Reference](#) (those Allen operators are different beast altogether because they can generate slave tokens etc.).

Constraint	Syntax	Implied Constraints
before	a before b	precedes(a.end, b.start);
after	a after b	precedes(b.end, a.start);
meets	a meets b	concurrent(a.end, b.start);
met_by	a met_by b	concurrent(a.start, b.end);
equal or equals	a equal b	concurrent(a.start, b.start); concurrent(a.end, b.end);
contains	a contains b	precedes(a.start, b.start); precedes(b.end, a.end);
contained_by	a contained_by b	precedes(b.start, a.start); precedes(a.end, b.end);
paralleled_by	a paralleled_by b	precedes(b.start, a.start); precedes(b.end, a.end);
parallels	a parallels b	precedes(a.start, b.start); precedes(a.end, b.end);
starts	a starts b	concurrent(a.start, b.start);
ends	a ends b	concurrent(a.end, b.end);
ends_after	a ends_after b	precedes(b.end, a.end);
ends_before	a ends_before b	precedes(a.end, b.end);
ends_after_start	a ends_after_start b	precedes(b.start, a.end);
starts_before_end	a starts_before_end b	precedes(a.start, b.end);
starts_during	a starts_during b	precedes(b.start, a.start); precedes(a.start, b.end);
contains_start	a contains_start b	precedes(a.start, b.start); precedes(b.start, a.end);
ends_during	a ends_during b	precedes(b.start, a.end); precedes(a.end, b.end);
contains_end	a contains_end b	precedes(a.start, b.end); precedes(b.end, a.end);
starts_after	a starts_after b	precedes(b.start, a.start);
starts_before	a starts_before b	precedes(a.start, b.start);

Comparison Constraints

These constraints enforce <, >, =, !=, etc.

Constraint	Syntax	Description	Variable Types
eq	eq(a,b,...)	a == b == ...	Comparable
neq	neq(a,b)	a != b	Comparable
lessThan	lessThan(a,b)	a < b	Numeric
leq	leq(a,b)	a <= b	Numeric
withinBounds	withinBounds(a,b,c)	a.lb >= b.lb, a.ub <= c.ub and b <= c	Comparable

Calculation Constraints

These constraints enforce that one variable (usually *a*) is constrained by a calculation done on the remaining variables.

Constraint	Syntax	Description
------------	--------	-------------

			Variable Types
distanceSquares	distanceSquares(a,b,c)	$c = \sqrt{a + b}$ if a and b are singleton	Numeric intervals
calcDistance	calcDistance(a,b,c,d,e)	a is the Euclidean distance between points (b,c) and (d,e)	Numeric intervals
sin	sin(a,b)	$a = \sin(b)$	Numeric intervals
addEq	addEq(a,b,c)	$a+b=c$	Numeric intervals
mulEq	mulEq(a,b,c)	$a*b=c$	Numeric intervals
addMulEq	addMulEq(a,b,c,d)	$a + (b*c)=d$	Numeric intervals
GreaterThanSum	GreaterThanSum(a,b,c,...)	$a > b + c + \dots$	Numeric
LessThanSum	LessThanSum(a,b,c,...)	$a < b + c + \dots$	Numeric
GreaterOrEqThanSum	GreaterOrEqThanSum(a,b,c,...)	$a \geq b + c + \dots$	Numeric
LessOrEqThanSum	LessOrEqThanSum(a,b,c,...)	$a \leq b + c + \dots$	Numeric
allDiff	allDiff(a,b,...)	Restrict all domains so the intersection of any pair of domains is empty	Comparable
EqualMaximum	EqualMax(a,b,...)	$a = \max(b, \dots)$	Numeric
EqualMinimum	EqualMin(a,b,...)	$a = \min(b, \dots)$	Numeric
EqualProduct	EqualProduct(a,b,c)	$a = b * c$	Numeric Intervals
EqualSum	EqualSum(a,b,c)	$a = b + c$	Numeric Intervals
CountZeros	CountZeros(a,b,...)	a is the count of the rest that can be zero	Numeric
CountNonZeros	CountNonZeros(a,b,...)	a is the count of the rest that can be non-zero	Numeric
diffSquare	diffSquare(a,b,c)	$c = (a - b)^2$ if a and b are singleton	Numeric intervals
card	card(a,b,...)	a must be greater than or equal the count of the other variables that are true	Numeric

Set Constraints

These constraints are likely to be used for set comparisons etc. Note, however, that many of the other constraints listed on this page could be used for sets (ie Comparable variables) just as these constraints could be applied to numeric domains as well.

Constraint	Syntax	Description	Variable Types
subsetOf	subsetOf(a,b)	a is a subset of b	Comparable
memberImPLY	memberImPLY(a,b,c,d)	If a is subset of b, then require that c is subset of d	a and b comparable, c and d comparable
Lock	Lock(a,b)		Comparable

Restrict a's domain to be contained in b's domain

Conditional and Test Constraints

- **Conditional Constraints:** The value of one variable (usually *a*) determines if a constraint among remaining variable must be enforced.
- **Test Constraints:** The value of one variable (usually *a*) can flip the constraint enforced among remaining variables.

Constraint	Syntax	Description	Variable Types
condAllDiff	condAllDiff(a,b,c,...)	If a, then $b \neq c \ \&\& \ b \neq d \ \&\& \ c \neq d \ \&\& \dots$ If not a, then $!(b \neq c \ \&\& \ b \neq d \ \&\& \ c \neq d \ \&\& \dots)$	a bool, the rest comparable
condEq	condEq(a,b,c,...)	If a, then $b == c \ \&\& \ b == d \ \&\& \ c == d \ \&\& \dots$ If not a, then $!(b == c \ \&\& \ b == d \ \&\& \ c == d \ \&\& \dots)$	a bool, the rest comparable
CondEqualSum	condEqualSum(a,b,c,d,...)	If a is true, then $b = c + d \dots$ If a is false, $b \neq c + d \dots$	Numeric intervals
testEQ	testEQ(a,b,c)	If a, then $b == c$ If not a, then $b \neq c$	a numeric, b and c comparable
testLEQ	testLEQ(a,b,c)	If a, then $b \leq c$ If not a, then $b > c$	Numeric
TestLessThan	TestLessThan(a,b,c)	If a, then $b < c$ If not a, then $b \geq c$	Numeric

Object Hierarchy Constraints

NOTE: Unlike most constraints on this page, which are imposed on variables, these constraints are imposed on objects and tokens.

These constraints are used to assert which object one or more tokens is contained by. Most often, the commonAncestor constraint is used to subgoal across timelines which must share a contained object in common.

Constraint	Syntax	Description	Variable Types
commonAncestor	commonAncestor(a,b,c)	a and b must be contained by the same object in c (or by c itself)	a, b, and c are all objects
hasAncestor	hasAncestor(a,b)	a must be contained by some object in b	a and b are objects

Here's a snippet of code showing the use of the commonAncestor constraint:

```
Instrument::TakeSample{
    contained_by(Navigator.At at);
    eq(at.location, rock);
    Rover rovers;
    commonAncestor(at.object, this.object, rovers);
}
```

This ensures that the Navigator object used for the At token and the Instrument object used for the TakeSample token are both contained in the same Rover (you wouldn't want one Rover to be in the right place and another to take the sample!).

Miscellaneous Constraints

Constraint	Syntax	Description	Variable Types
UNARY	NA	Restrict a variable's domain: given a variable and a domain, intersect them. Note that this constraint is only available internally and not exposed in nddl.	Anything
neg	neg(a,b)	$a \geq 0, b \leq 0, a+b=0$	Numeric intervals
or	or(a,b,...)	At least one of the variables must be true	Numeric
absVal	absVal(a,b)	$a.lb \geq 0, a.ub = \max(\text{abs}(b.lb), \text{abs}(b.ub)), b.lb \geq -a.lb, b.ub \leq a.ub$	Numeric

Deprecated Constraint Names

Historically, other names have been used for the above constraints. The following lists alternatives that are now deprecated:

Constraint Kept	Deprecated Equivalents
eq	Equal, asame, fasame
addEq	AddEqual, addeq
mulEq	multEq, MultEqual
addMulEq	AddMultEqual, addmuleq
allDiff	AllDiff, adiff, fadiff, fneq
card	Cardinality
condEq	CondAllSame, condeq, condasame
CountNonZeros	cardeq
EqualMaximum	fallmax
EqualMinimum	fallmin
EqualProduct	product
EqualSum	sum
leq	LessThanEqual
lessThan	LessThan, lt
LessOrEqThanSum	leqsum
TestLessThan	condlt
withinBounds	WithinBounds
testEQ	TestEqual
testLEQ	condleq
TestLessThan	condlt
memberImply	MemberImply
neq	NotEqual
or	for, Or
subsetOf	SubsetOf, singleton

temporalDistance temporaldistance